# Deterministic methods for finding elements of large multiplicative order

David Harvey (UNSW Sydney) & Markus Hittmeir (NORCE Research)

ANTS XVII (2026), Groningen

# The large order problem

Notation: $\mathbf{Z}_N := \mathbf{Z}/N\mathbf{Z}$.

**The large order problem**

Given positive integers $N$ and $D$, find (if possible) an element $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$.

# The large order problem

Notation: $\mathbf{Z}_N := \mathbf{Z}/N\mathbf{Z}$.

**The large order problem**

Given positive integers $N$ and $D$, find (if possible) an element $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$.

A solution exists if and only if $D < \lambda(N)$ where

$$\begin{aligned}\lambda(N) &= \text{exponent of the group } \mathbf{Z}_N^* \\ &= \text{maximum order of an element of } \mathbf{Z}_N^*.\end{aligned}$$

$\lambda(N)$ is called the **Carmichael function**.

# The large order problem

Notation: $\mathbf{Z}_N := \mathbf{Z}/N\mathbf{Z}$.

**The large order problem**

Given positive integers $N$ and $D$, find (if possible) an element $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$.

A solution exists if and only if $D < \lambda(N)$ where

$$\begin{aligned}\lambda(N) &= \text{exponent of the group } \mathbf{Z}_N^* \\ &= \text{maximum order of an element of } \mathbf{Z}_N^*.\end{aligned}$$

$\lambda(N)$ is called the **Carmichael function**.

Note that $\lambda(N)$ can be substantially smaller than $N$.

Example: if $N = 289442201 = 401 \times 601 \times 1201$ then

$$\lambda(N) = \mathrm{LCM}(400, 600, 1200) = 1200 \lll N.$$

# The large order problem

Solving the large order problem is an important step in several recent algorithms for **rigorous and deterministic integer factorisation**.

But first let's briefly discuss **heuristic** and/or **randomised** approaches to the large order problem.

# Monte Carlo methods

Suppose that our goal is merely to find $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$ with **high probability**.

# Monte Carlo methods

Suppose that our goal is merely to find $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$ with **high probability**.

Here is the algorithm:

1. Choose random $\alpha \in \mathbf{Z}_N^*$.
2. Return $\alpha$.

# Monte Carlo methods

Suppose that our goal is merely to find $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$ with **high probability**.

Here is the algorithm:

1. Choose random $\alpha \in \mathbf{Z}_N^*$.
2. Return $\alpha$.

Success probability depends on the density of elements of order $> D$.

This algorithm can be quite good when $D$ is small!

# Monte Carlo methods

If we know the prime factorisation $N = p_1^{e_1} \cdots p_k^{e_k}$, then we can boost the success probability considerably.

(This includes the case that $N$ is prime.)

# Monte Carlo methods

If we know the prime factorisation $N = p_1^{e_1} \cdots p_k^{e_k}$, then we can boost the success probability considerably.

(This includes the case that $N$ is prime.)

First compute $\phi(N)$ and find all of its "small" prime divisors $q$.

Given any $\alpha \in \mathbf{Z}_N^*$, we can easily determine the $q$-part of $\mathrm{ord}_N(\alpha)$ for such primes $q$, by computing $\alpha^{\phi(N)/q^j} \pmod{p_i^{e_i}}$ for various $i$ and $j$.

# Monte Carlo methods

If we know the prime factorisation $N = p_1^{e_1} \cdots p_k^{e_k}$, then we can boost the success probability considerably.

(This includes the case that $N$ is prime.)

First compute $\phi(N)$ and find all of its "small" prime divisors $q$.

Given any $\alpha \in \mathbf{Z}_N^*$, we can easily determine the $q$-part of $\mathrm{ord}_N(\alpha)$ for such primes $q$, by computing $\alpha^{\phi(N)/q^j} \pmod{p_i^{e_i}}$ for various $i$ and $j$.

By sampling repeatedly, we easily find $\alpha \in \mathbf{Z}_N^*$ with maximal $q$-power order for all such $q$.

Such $\alpha$ are overwhelmingly likely to have maximal $q$-power order for the remaining (unknown but large) prime divisors $q \mid \phi(N)$.

# Las Vegas methods

Suppose now that we want an element $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$ **with certainty**.

Then we need a way to actually compute the order of elements, or otherwise certify that the order is large.

# Las Vegas methods

Suppose now that we want an element $\alpha \in \mathbf{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$ **with certainty**.

Then we need a way to actually compute the order of elements, or otherwise certify that the order is large.

If we know the prime factorisation of **both** $N$ and $\phi(N)$, then we can compute orders in (deterministic) polynomial time.

In this situation we can sample repeatedly until success, so the Monte Carlo algorithm is promoted to Las Vegas.

# Las Vegas methods

Example: if $N$ is prime and we can factor $\phi(N) = N - 1$, then we can find a primitive root modulo $N$ in randomised polynomial time.

# Las Vegas methods

Example: if $N$ is prime and we can factor $\phi(N) = N - 1$, then we can find a primitive root modulo $N$ in randomised polynomial time.

If we insist on deterministic algorithms, then we may be able to substitute heuristic assumptions for randomness.

For example, under suitable GRH it is known that for prime $N$ there exists a primitive root modulo $N$ less than $(\log N)^{O(1)}$.

So here we get a **deterministic** polynomial time algorithm for finding a primitive root — again assuming the factorisation of $N - 1$ is known.

# Factorisation is the bottleneck

Bottom line: in all of these (heuristic and/or probabilistic) methods, the bottleneck is **integer factorisation**.

So in general we probably can't do better than the number field sieve, i.e., $\exp((C + o(1))(\log N)^{1/3}(\log\log N)^{1/3})$.

# From now on everything is rigorous and deterministic

# Why do we care about the large order problem?

Motivation: **Pollard's approach** to deterministic integer factorisation (1974).

Suppose we want to factor $N = pq$ where $p < N^{1/2}$.

Let $m := \lceil N^{1/4} \rceil$ and write $p - 1 = im + j$ with $0 \leqslant i, j < m$.

Choose $\alpha \in \mathbf{Z}_N^*$. Since $\alpha^{p-1} \equiv 1 \pmod{p}$ we get $\alpha^{im} \equiv \alpha^{-j} \pmod{p}$.

Compute the polynomial $f(x) = (x-1)(x-\alpha^{-1}) \cdots (x-\alpha^{-m+1}) \in \mathbf{Z}_N[x]$ and evaluate it at $\alpha^m, \alpha^{2m}, \ldots, \alpha^{m^2}$. This can be done in time $N^{1/4+o(1)}$ using fast polynomial arithmetic.

After some further steps (omitted), we can recover the divisor $p$ from the term $\alpha^{im} - \alpha^{-j}$,

# Why do we care about the large order problem?

Motivation: **Pollard's approach** to deterministic integer factorisation (1974).

Suppose we want to factor $N = pq$ where $p < N^{1/2}$.

Let $m := \lceil N^{1/4} \rceil$ and write $p - 1 = im + j$ with $0 \leqslant i, j < m$.

Choose $\alpha \in \mathbf{Z}_N^*$. Since $\alpha^{p-1} \equiv 1 \pmod{p}$ we get $\alpha^{im} \equiv \alpha^{-j} \pmod{p}$.

Compute the polynomial $f(x) = (x-1)(x-\alpha^{-1}) \cdots (x-\alpha^{-m+1}) \in \mathbf{Z}_N[x]$ and evaluate it at $\alpha^m, \alpha^{2m}, \ldots, \alpha^{m^2}$. This can be done in time $N^{1/4+o(1)}$ using fast polynomial arithmetic.

After some further steps (omitted), we can recover the divisor $p$ from the term $\alpha^{im} - \alpha^{-j}$, provided that we chose $\alpha$ carefully so that $\mathrm{ord}_N(\alpha) > N^{1/2}$.

# Why do we care about the large order problem?

Pollard's algorithm was subsequently improved:

- Hittmeir (2021): achieved $N^{2/9+o(1)}$ (by combining with ideas of Lehman 1974)

# Why do we care about the large order problem?

Pollard's algorithm was subsequently improved:

- Hittmeir (2021): achieved $N^{2/9+o(1)}$ (by combining with ideas of Lehman 1974)
- H. (2021): achieved $N^{1/5+o(1)}$ (same idea as Hittmeir but trying harder)

# Why do we care about the large order problem?

Pollard's algorithm was subsequently improved:

- Hittmeir (2021): achieved $N^{2/9+o(1)}$ (by combining with ideas of Lehman 1974)
- H. (2021): achieved $N^{1/5+o(1)}$ (same idea as Hittmeir but trying harder)
- H.–Hittmeir (2022): achieved

$$O\left(\frac{N^{1/5}(\log N)^{16/5}}{(\log\log N)^{3/5}}\right)$$

  bit operations (current record).

The $N^{1/5+o(1)}$ versions require access to an element of order at least $N^{2/5}$.

# Hittmeir's theorem

Fortunately, for the $N^{1/5+o(1)}$ algorithms we already had available the following result.

**Theorem (Hittmeir, 2018)**

There is an algorithm that takes as input integers $N \geqslant 2$ and $D \in [N^{2/5}, N]$, and outputs

(i) some $\alpha \in \mathbf{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > D$, or

(ii) a non-trivial factor of $N$, or

(iii) "$N$ is prime".

It runs in $O\left(\dfrac{D^{1/2}\log^2 N}{(\log\log D)^{1/2}}\right)$ bit operations.

The running time is the same as Sutherland's optimisation (2007) of Shanks' baby-step/giant-step method (1971) for searching for the order of a **single** element up to $D$.

# Hittmeir's theorem

Hittmeir's result was subsequently improved:

- Gao, Feng, Hu and Pan (2025): weakened hypothesis to $D \geqslant N^{1/4+o(1)}$.
- Oznovich and Volk (2026): weakened further to $D \geqslant N^{1/6}$.
- Nir ($>$ 2026?): weakened even further to $D > \exp(\sqrt{2 \log N \log\log N})$.

# New result

**Theorem (H.–Hittmeir, 2026)**

There is an algorithm that takes as input integers $N \geqslant 3$ and $D \in [1, N-1)$, and outputs

(i) some $\alpha \in \mathbf{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > D$, or

(ii) a non-trivial factor of $N$.

It runs in $O\left(\dfrac{D^{1/2} \log D \log N}{(\log\log D)^{1/2}}\right)$ bit operations.

Key differences with previous results:

- Hypothesis on $D$ completely removed.
- Never returns "$N$ is prime". So if $N$ is prime then it *always* returns $\alpha$ with $\mathrm{ord}_N(\alpha) > D$.
- Complexity slightly improved: $\log^2 N \Longrightarrow \log D \log N$ (thanks to anonymous referee, possibly in the audience).

# The algorithm — preliminary steps

- If $N$ is small, use a lookup table.
- If $N$ is even, return divisor 2.
- If $D < \log_2 N$, return $\alpha = 2$ (which has order $> D$).

So from now on we can assume $N$ and $D$ are both large.

# The algorithm — strategy for the main loop

We maintain an element $\alpha \in \mathbf{Z}_N^*$ of known order $M$, and the factorisation of $M$.

The initial values are $\alpha := 1$ and $M := 1$.

We iterate over $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$.

For each $\beta$ we use BSGS to search for the order of $\beta$ up to $D$.

If $\mathrm{ord}_N(\beta) > D$ then we already win.

Otherwise, we get information that allows us to improve our choice of $\alpha$.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.
4. Compute the prime factorisation of $m$.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.
4. Compute the prime factorisation of $m$.
5. For each prime $r \mid m$, check if $\gcd(\beta^{m/r} - 1, N) = 1$.
   If not, then nontrivial divisor found, we win.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.
4. Compute the prime factorisation of $m$.
5. For each prime $r \mid m$, check if $\gcd(\beta^{m/r} - 1, N) = 1$.
   If not, then nontrivial divisor found, we win.
6. Combine $\alpha$ and $\beta$ to obtain new element $\alpha'$ of known order $M' := \mathrm{LCM}(M, m)$ (with known factorisation).

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.
4. Compute the prime factorisation of $m$.
5. For each prime $r \mid m$, check if $\gcd(\beta^{m/r} - 1, N) = 1$.
   If not, then nontrivial divisor found, we win.
6. Combine $\alpha$ and $\beta$ to obtain new element $\alpha'$ of known order $M' := \mathrm{LCM}(M, m)$ (with known factorisation).
7. If $M' > D$ then $\mathrm{ord}_N(\alpha') > D$, we win.

# The algorithm — main loop

Now in more detail. For $\beta = 2, 3, \ldots, \lceil D^{1/3} \rceil$:

1. If $\beta \mid N$ then nontrivial divisor found, we win.
2. If $\beta^M \equiv 1 \pmod{N}$ then skip to next value of $\beta$.
3. Use BSGS to search for $m := \mathrm{ord}_N(\beta)$ up to $D$. If this fails then $\mathrm{ord}_N(\beta) > D$, we win.
4. Compute the prime factorisation of $m$.
5. For each prime $r \mid m$, check if $\gcd(\beta^{m/r} - 1, N) = 1$.
   If not, then nontrivial divisor found, we win.
6. Combine $\alpha$ and $\beta$ to obtain new element $\alpha'$ of known order $M' := \mathrm{LCM}(M, m)$ (with known factorisation).
7. If $M' > D$ then $\mathrm{ord}_N(\alpha') > D$, we win.
8. Set $\alpha := \alpha'$ and $M := M'$ for next iteration.

# The algorithm — main loop

The complexity of the main loop is dominated by the BSGS searches.

The key question is: what happens if we still haven't won after checking all $\beta \leqslant \lceil D^{1/3} \rceil$?

# The algorithm — main loop

The complexity of the main loop is dominated by the BSGS searches.

The key question is: what happens if we still haven't won after checking all $\beta \leqslant \lceil D^{1/3} \rceil$?

Let $p$ be any prime divisor of $N$.

- The GCD checks ensure that always $\mathrm{ord}_p(\beta) = m$.
- Since $M$ is the LCM of these $m$'s, we get $M \mid p - 1$, i.e., $p \equiv 1 \pmod{M}$.

# The algorithm — main loop

But also:

- We know that $\beta^M \equiv 1 \pmod{N}$ for all $\beta \leqslant \lceil D^{1/3} \rceil$.
- Hence $x^M \equiv 1 \pmod{p}$ for all $D^{1/3}$-smooth integers $x < p$.
- But $x^M \equiv 1 \pmod{p}$ has at most $M$ solutions!

# The algorithm — main loop

But also:

- We know that $\beta^M \equiv 1 \pmod{N}$ for all $\beta \leqslant \lceil D^{1/3} \rceil$.
- Hence $x^M \equiv 1 \pmod{p}$ for all $D^{1/3}$-smooth integers $x < p$.
- But $x^M \equiv 1 \pmod{p}$ has at most $M$ solutions!

Combining with known lower bounds for the smooth integer counting function $\Psi(x,y)$ (e.g. Konyagin–Pomerance 1997), we get a nontrivial upper bound for $p$:

$$p \ll M \cdot (\log 2M)^{(\log 2M)/(-1+\log D^{1/3})}.$$

(This smoothness argument is the key new ingredient compared to previous papers.)

# The algorithm — main loop

But also:

- We know that $\beta^M \equiv 1 \pmod{N}$ for all $\beta \leqslant \lceil D^{1/3} \rceil$.
- Hence $x^M \equiv 1 \pmod{p}$ for all $D^{1/3}$-smooth integers $x < p$.
- But $x^M \equiv 1 \pmod{p}$ has at most $M$ solutions!

Combining with known lower bounds for the smooth integer counting function $\Psi(x,y)$ (e.g. Konyagin–Pomerance 1997), we get a nontrivial upper bound for $p$:

$$p \ll M \cdot (\log 2M)^{(\log 2M)/(-1+\log D^{1/3})}.$$

(This smoothness argument is the key new ingredient compared to previous papers.)

All that remains is to check for divisors $p = M+1, 2M+1, 3M+1, \ldots$ up to this bound.

This is very fast — in fact runs in polynomial time.

# Thanks for your attention!